In [56]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler,StandardScaler,RobustScaler #use std scaler only when the data is norma
from sklearn.preprocessing import PowerTransformer
from warnings import filterwarnings
filterwarnings('ignore')
import matplotlib.cm as cm
from statsmodels.graphics.gofplots import qqplot
from statsmodels.api import add_constant

import statsmodels
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.tsa.api as smt

from scipy import stats

np.set_printoptions(suppress = True)

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.cluster import DBSCAN

from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cophenet

from sklearn.model_selection import train_test_split
from numpy.linalg import eig
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
```

In [2]:
```
df = pd.read_csv('student_evaluation_reduced (1).csv')
df
```

Out[2]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28
0	3	8	1	0	1	5	5	5	5	5	...	5	5	5	5	5	5	5	5	5	5
1	3	8	1	4	3	3	3	3	3	3	...	3	3	3	3	3	3	3	3	3	3
2	3	13	1	3	3	3	3	3	3	3	...	3	3	3	3	3	3	3	3	3	3
3	3	5	1	0	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1
4	3	3	1	3	4	2	2	4	2	3	...	4	4	3	4	3	3	3	3	2	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	3	13	1	0	1	2	2	2	2	2	...	3	2	2	2	2	2	2	2	2	3
496	2	1	1	3	4	4	4	4	5	4	...	5	3	4	4	5	4	4	4	4	4
497	1	10	1	1	3	4	4	1	3	5	...	3	3	3	3	3	3	3	3	3	3
498	3	3	1	3	1	5	5	5	5	5	...	5	5	5	5	5	5	5	5	5	5
499	1	7	3	1	4	3	3	3	3	3	...	3	3	3	3	3	3	3	3	3	3

500 rows × 33 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instr       500 non-null    int64
 1   class       500 non-null    int64
 2   nb.repeat   500 non-null    int64
 3   attendance  500 non-null    int64
 4   difficulty  500 non-null    int64
 5   Q1          500 non-null    int64
 6   Q2          500 non-null    int64
 7   Q3          500 non-null    int64
 8   Q4          500 non-null    int64
 9   Q5          500 non-null    int64
 10  Q6          500 non-null    int64
 11  Q7          500 non-null    int64
 12  Q8          500 non-null    int64
 13  Q9          500 non-null    int64
```

In [4]:
```
round(df.describe(),3)
```

Out[4]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19	Q20	Q21	Q22	Q
count	500.000	500.000	500.000	500.000	500.000	500.00	500.000	500.000	500.000	500.000	...	500.000	500.000	500.000	500.000	500.
mean	2.532	7.374	1.214	1.530	2.742	2.98	3.134	3.222	3.124	3.144	...	3.344	3.344	3.362	3.360	3.
std	0.694	3.765	0.534	1.488	1.359	1.38	1.314	1.301	1.345	1.322	...	1.307	1.318	1.297	1.303	1.
min	1.000	1.000	1.000	0.000	1.000	1.00	1.000	1.000	1.000	1.000	...	1.000	1.000	1.000	1.000	1.
25%	2.000	4.000	1.000	0.000	1.000	2.00	2.000	2.000	2.000	2.000	...	3.000	3.000	3.000	3.000	2.
50%	3.000	7.000	1.000	1.000	3.000	3.00	3.000	3.000	3.000	3.000	...	3.000	3.000	3.000	4.000	3.
75%	3.000	11.000	1.000	3.000	4.000	4.00	4.000	4.000	4.000	4.000	...	4.000	4.000	4.000	4.000	4.
max	3.000	13.000	3.000	4.000	5.000	5.00	5.000	5.000	5.000	5.000	...	5.000	5.000	5.000	5.000	5.

8 rows × 33 columns

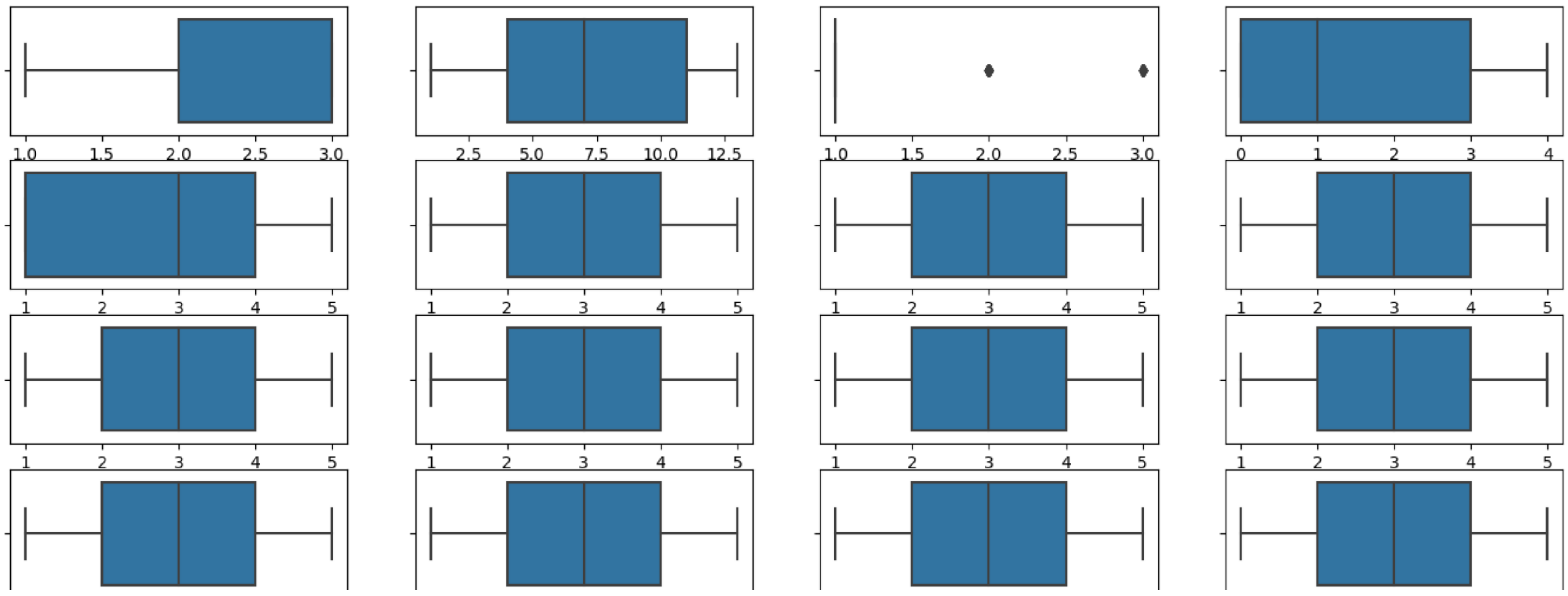
In [5]:
```
df.isna().sum()
```

Out[5]:
```
instr         0
class         0
nb.repeat     0
attendance    0
difficulty    0
Q1            0
Q2            0
Q3            0
Q4            0
Q5            0
Q6            0
Q7            0
Q8            0
Q9            0
Q10           0
Q11           0
Q12           0
Q13           0
Q14           0
```

In [7]:
```
df1 = df.columns
df1
```

Out[7]:
```
Index(['instr', 'class', 'nb.repeat', 'attendance', 'difficulty', 'Q1', 'Q2',
       'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
       'Q14', 'Q15', 'Q16', 'Q17', 'Q18', 'Q19', 'Q20', 'Q21', 'Q22', 'Q23',
       'Q24', 'Q25', 'Q26', 'Q27', 'Q28'],
      dtype='object')
```

In [9]:
```python
t=1
plt.figure(figsize = (17,15))
for i in df1:
    plt.subplot(9,4,t)
    sns.boxplot(df[i])
    t+=1
plt.show()
```



In [108]:
```python
df.skew()
```

Out[108]:
```
instr         -1.161289
class          0.086780
nb.repeat      2.447698
attendance     0.348447
difficulty     0.008813
Q1            -0.042147
Q2            -0.206242
Q3            -0.307886
Q4            -0.202412
Q5            -0.193517
Q6            -0.270950
Q7            -0.172344
Q8            -0.108285
Q9            -0.268426
Q10           -0.229773
Q11           -0.276114
Q12           -0.130354
Q13           -0.378527
Q14           -0.447448
```

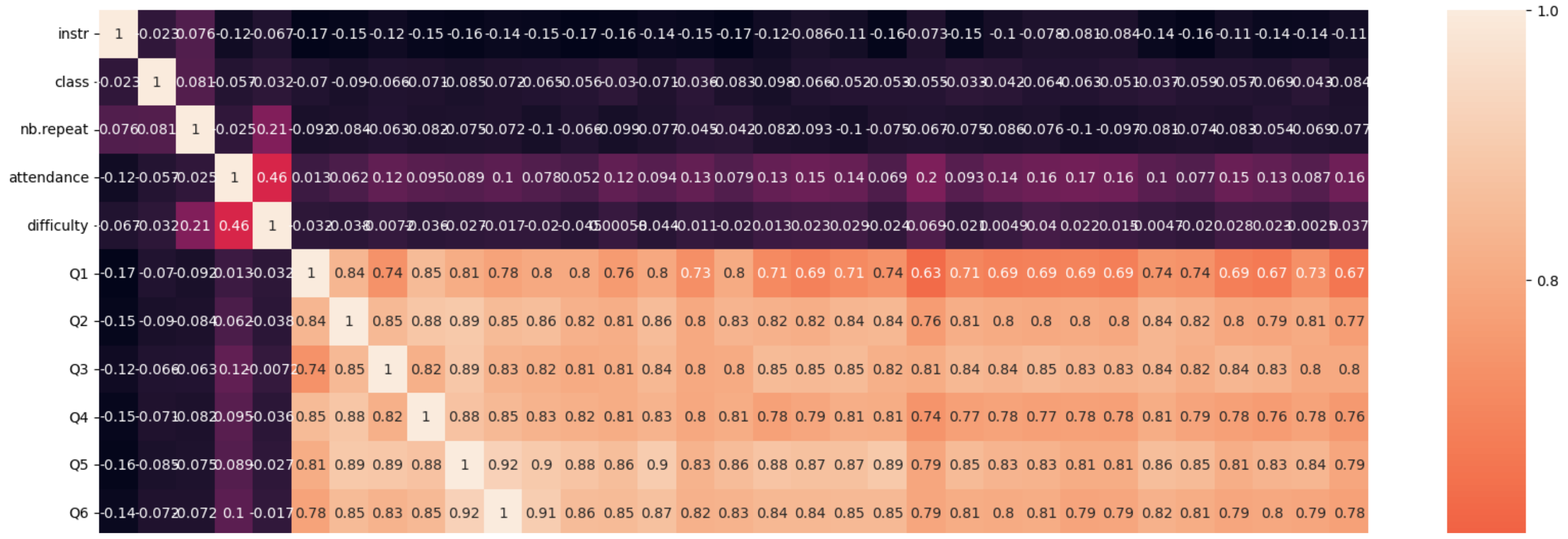
In [10]:
```python
df.corr()
```

Out[10]:

	instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	...	Q19	Q2
instr	1.000000	-0.023370	0.076143	-0.118317	-0.066634	-0.170904	-0.152960	-0.124389	-0.152383	-0.157924	...	-0.104894	-0.07776
class	-0.023370	1.000000	0.080765	-0.056917	-0.031623	-0.069923	-0.090322	-0.066082	-0.070919	-0.084938	...	-0.041658	-0.06352
nb.repeat	0.076143	0.080765	1.000000	-0.024509	0.208909	-0.092166	-0.083825	-0.062802	-0.081736	-0.075036	...	-0.085625	-0.07638
attendance	-0.118317	-0.056917	-0.024509	1.000000	0.456263	0.012985	0.061991	0.123399	0.095300	0.088504	...	0.138929	0.15619
difficulty	-0.066634	-0.031623	0.208909	0.456263	1.000000	-0.031614	-0.037823	-0.007212	-0.036190	-0.027249	...	0.004936	0.03957
Q1	-0.170904	-0.069923	-0.092166	0.012985	-0.031614	1.000000	0.835876	0.738357	0.853620	0.811582	...	0.688258	0.68929
Q2	-0.152960	-0.090322	-0.083825	0.061991	-0.037823	0.835876	1.000000	0.848773	0.881800	0.893332	...	0.803523	0.79932
Q3	-0.124389	-0.066082	-0.062802	0.123399	-0.007212	0.738357	0.848773	1.000000	0.818276	0.888248	...	0.835292	0.84722
Q4	-0.152383	-0.070919	-0.081736	0.095300	-0.036190	0.853620	0.881800	0.818276	1.000000	0.875099	...	0.781639	0.77417
Q5	-0.157924	-0.084938	-0.075036	0.088504	-0.027249	0.811582	0.893332	0.888248	0.875099	1.000000	...	0.831881	0.82743
Q6	-0.139502	-0.072400	-0.071724	0.104051	-0.017220	0.783685	0.852389	0.829586	0.845379	0.918232	...	0.804755	0.80863

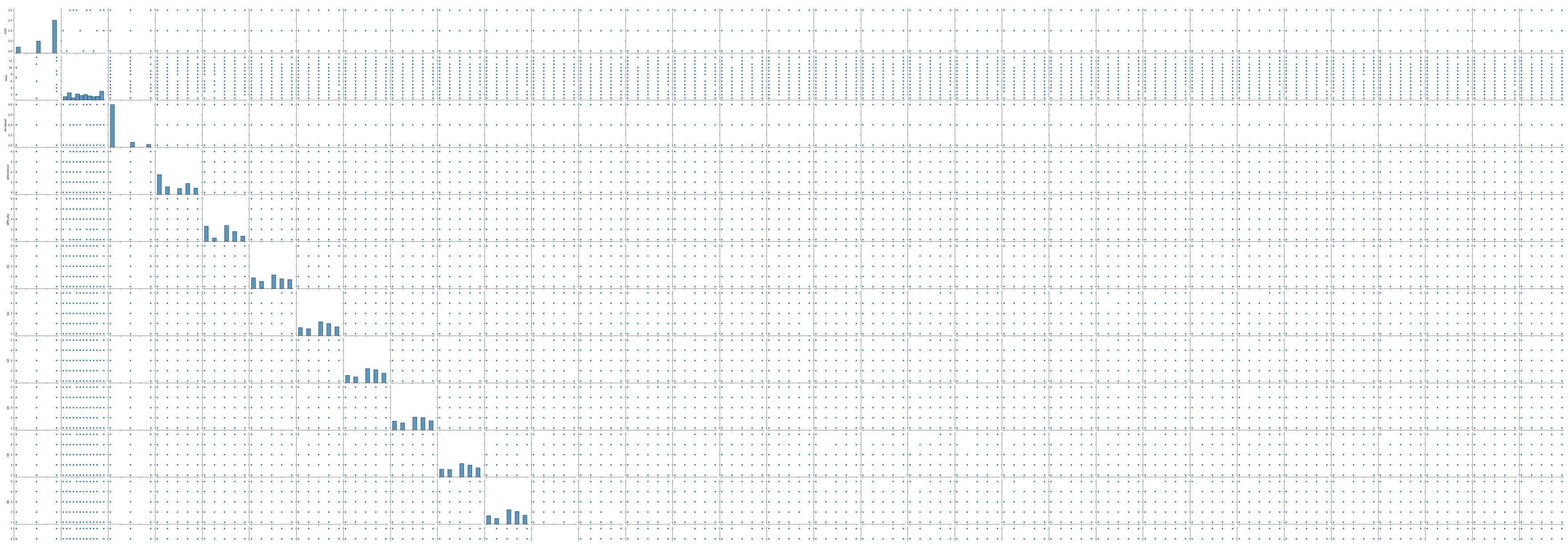
In [11]:
```python
plt.figure(figsize=(20, 20))
sns.heatmap(df.corr(), annot = True)
```

Out[11]: <AxesSubplot:>



In [12]:
```python
#sns.pairplot(df)
```

Out[12]: <seaborn.axisgrid.PairGrid at 0x22757169d90>



In [13]:
```python
#3
cov_mat = np.cov(df.T)

# as 'cov_mat' is a numpy array, select first five observations with [0:5]
print(cov_mat[0:5])
```

```
[[ 0.48193988 -0.06109018  0.02820842 -0.12220441 -0.06286974 -0.16368737
  -0.13956713 -0.11232866 -0.14225251 -0.1448978  -0.12737475 -0.13653707
  -0.1572986  -0.14336673 -0.12349499 -0.1368016  -0.15529459 -0.10636473
  -0.07716232 -0.09950301 -0.14908216 -0.06638076 -0.13381964 -0.0951984
  -0.0711503  -0.07272946 -0.07567134 -0.12891383 -0.15112625 -0.10334269
  -0.12776754 -0.1301483  -0.10449699]
 [-0.06109018 14.17848096  0.16228858 -0.31885772 -0.16183166 -0.36324649
  -0.44701002 -0.32367535 -0.35909419 -0.4227014  -0.35855711 -0.3268016
  -0.2817515  -0.14735471 -0.34803206 -0.18266132 -0.41201202 -0.48192786
  -0.31929459 -0.25493788 -0.26581964 -0.26937475 -0.16459319 -0.20506613
  -0.31528657 -0.31000802 -0.24913828 -0.18215631 -0.30066934 -0.28271743
  -0.33816032 -0.2172505  -0.42045691]
 [ 0.02820842  0.16228858  0.28477355 -0.01945892  0.15151503 -0.06785571
  -0.05879359 -0.04359519 -0.05865331 -0.05292184 -0.05034068 -0.07197194
  -0.04736273 -0.0687976  -0.0532024  -0.03171944 -0.02932665 -0.05731864
  -0.06374349 -0.0710501  -0.05347495 -0.04668938 -0.05260922 -0.05973547
  -0.05372345 -0.07161122 -0.06717435 -0.05662525 -0.05305411 -0.05858918
  -0.036998   -0.04947495 -0.05428457]
 [-0.12220441 -0.31885772 -0.01945892  2.21352705  0.92258517  0.02665331
```

In [14]:
```python
eig_val, eig_vec = np.linalg.eig(cov_mat)

print('Eigenvalues:','\n','\n', eig_val,"\n")

print('Eigenvectors:','\n','\n',eig_vec,'\n')
```

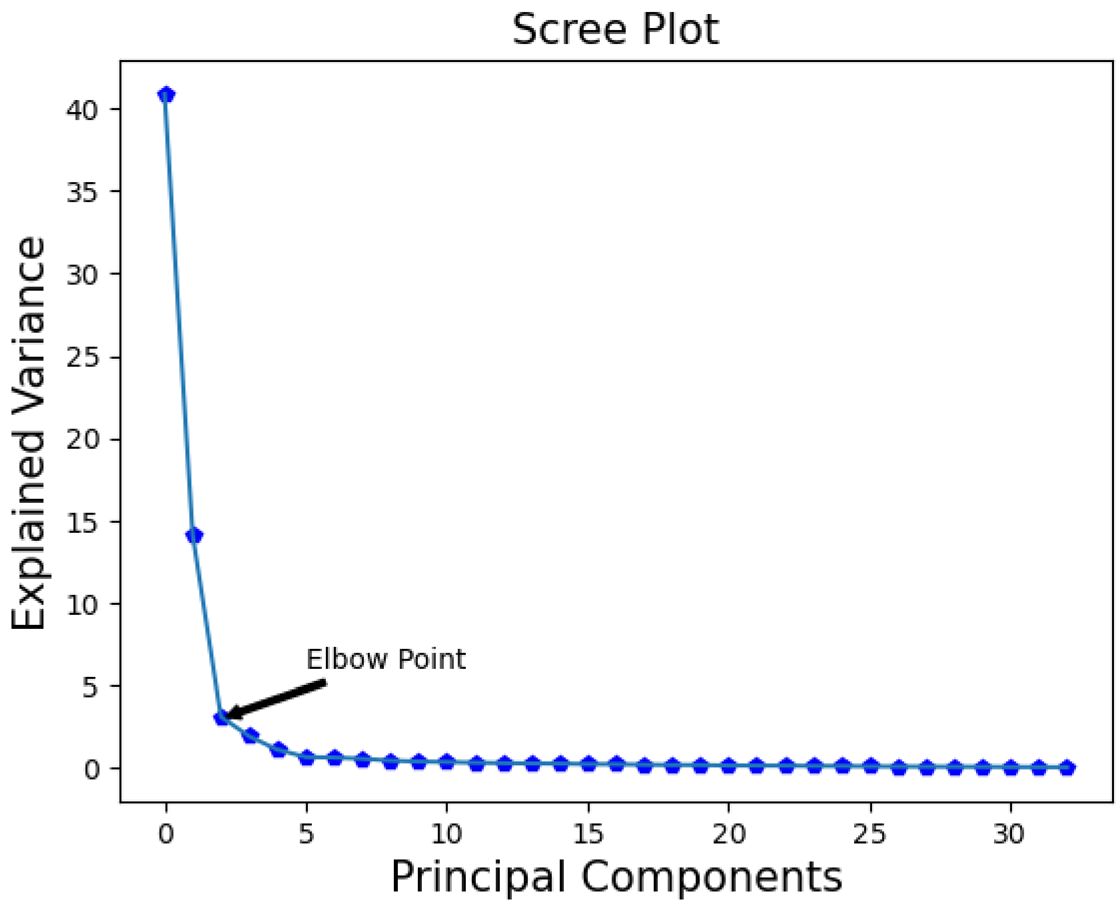
```
Eigenvalues:

 [40.8817616  14.10595168  3.11863913  1.93425971  1.11758087  0.66853536
  0.6372752   0.56522642  0.45319838  0.39400576  0.3644077   0.32626336
  0.29905634  0.27865971  0.26782365  0.24016007  0.23125354  0.04696903
  0.20339455  0.18468878  0.1860792   0.05904627  0.06432788  0.07668342
  0.08068443  0.08836345  0.09709278  0.10005827  0.12343929  0.12616951
  0.15230792  0.14858998  0.14263192]

Eigenvectors:

 [[ 0.01573661  0.0070146   0.0242403  ... -0.00885556  0.05299544
  -0.00859442]
 [ 0.06068539 -0.99714175 -0.02229391 ... -0.00029104 -0.01105689
   0.00362151]
 [ 0.00742362 -0.01026389 -0.02958134 ...  0.09497082  0.07093429
  -0.02535958]
 ...
 [-0.18736572 -0.00918177 -0.04415304 ... -0.12628904 -0.12660605
```

In [15]:
```python
# create a list of eigenvalues
eig_val = list(eig_val)

# 'sort(reverse = True)' will sort the eigenvalues in the descending order
eig_val.sort(reverse = True)

# print the sorted list
print(eig_val)
```

```
[40.88176160469996, 14.105951675873316, 3.118639128970837, 1.93425971494834, 1.1175808718301066, 0.6685353562147048, 0.6372751955247451, 0.5652264237808393, 0.4531983832978018, 0.39400575824459955, 0.364407697734742, 0.32626336243796816, 0.299056344673724, 0.2786597085119883, 0.26782365222512594, 0.2401600678023966, 0.2312535447555718, 0.20339455374206933, 0.18607919593248723, 0.1846887839790378, 0.15230791687013553, 0.14858997676709662, 0.14263192311471656, 0.1261695142399705, 0.12343929031820433, 0.1000582665914212, 0.09709277868218312, 0.0883634450000908, 0.08068443204694195, 0.07668341821635118, 0.06432787709401604, 0.05904627357123884, 0.0469690326480002]
```

a) **Kaiser criterion** : This criterion considers the number of pricipal components for which the eigenvalue is greater than 1. This criterion suffers a drawback of selecting more number of components as the eigenvalues very close to 1 may not contribute significantly in explaining the variation in the data. Here the first five eigenvalues are greater than 1. Thus we can consider 5 principal components using kaiser criterion.

In [26]:
```python
#b
plt.plot(eig_val,'bp')

plt.plot(eig_val)

plt.title('Scree Plot', fontsize = 15)
plt.xlabel('Principal Components', fontsize = 15)
plt.ylabel('Explained Variance', fontsize = 15)

plt.annotate(text='Elbow Point', xy=(2,3), xytext=(5,6), arrowprops=dict(facecolor='black', arrowstyle = 'simple'))

plt.show()
```



Interpretation: It can be observed that, after the elbow point, the principal components do not contribute much to the variance in the data. The Kaiser criterion considers the number of principal components as 5, but the scree plot shows that only first three components explains most of the variation.

In [24]:
```python
#c Percentage of Explained Variation

percent_var = []

for i in eig_val:
    variation = (i/sum(eig_val))*100
    percent_var.append(variation)

percent_var
```

Out[24]:
```
[60.32909594581732,
 20.816111602269824,
 4.602166634874946,
 2.8543814000870302,
 1.6492108215830268,
 0.9865556684722538,
 0.9404251408354645,
 0.8341029792479687,
 0.6687835277949257,
 0.5814331442510599,
 0.5377553729853515,
 0.4814658890301411,
 0.4413165725459428,
 0.41121731625969177,
 0.3952265797125358,
 0.354403509146707,
 0.3412601791544458,
 0.30014874765453625,
 0.2745965248141599,
```

Interpretation: It can be seen that the first principal component explains 60.32% variation in the data.

In [38]:
```
np.cumsum(percent_var)
```

Out[38]:
```
array([ 60.32909595,  81.14520755,  85.74737418,  88.60175558,
        90.2509664 ,  91.23752207,  92.17794721,  93.01205019,
        93.68083372,  94.26226687,  94.80002224,  95.28148813,
        95.7228047 ,  96.13402202,  96.5292486 ,  96.8836521 ,
        97.22491228,  97.52506103,  97.79965756,  98.07220225,
        98.2969626 ,  98.5162364 ,  98.72671791,  98.91290589,
        99.09506489,  99.24272058,  99.38600011,  99.51639779,
        99.63546357,  99.74862506,  99.84355352,  99.93068794,
       100.        ])
```

In [39]:
```
pca = PCA(n_components = 5, random_state = 10)

components = pca.fit_transform(df)
```

In [40]:
```
df_pca = pd.DataFrame(data = components, columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])

df_pca.head()
```

Out[40]:

	PC1	PC2	PC3	PC4	PC5
0	-9.262208	-1.246360	-2.567580	0.266860	0.200472
1	1.168775	-0.483509	1.756509	-1.088134	1.346846
2	1.502789	-5.490850	1.147391	-0.939926	0.756805
3	10.736984	2.965870	-2.411270	-0.465814	0.747689
4	-0.252974	4.416256	2.389900	1.076906	0.145288

In [37]:
```
df_pca.shape
```

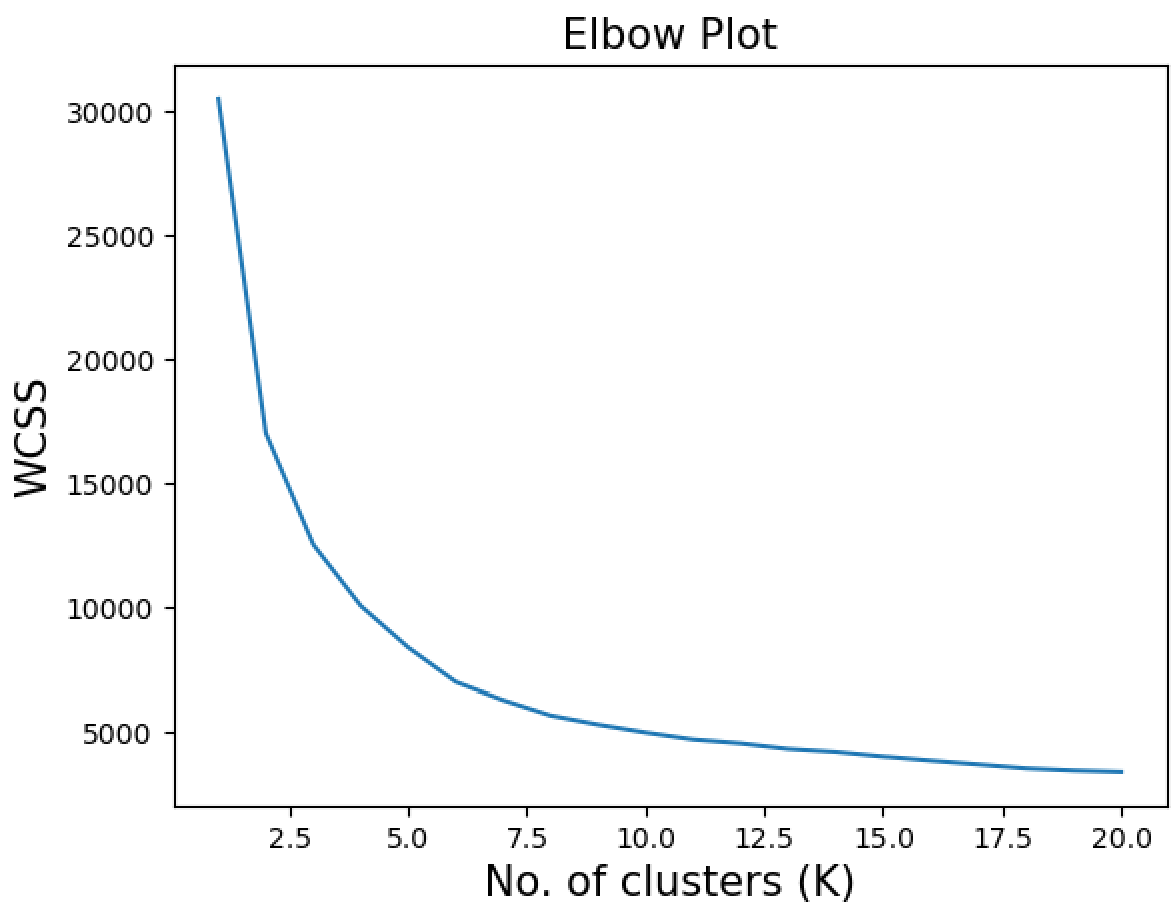
Out[37]:
```
(500, 5)
```

Interpretation: In the above step, we obtained the data with reduced dimensions. The new dataset has 500 observations and 5 columns, i.e. we have decreased the number of features from 33 to 5.

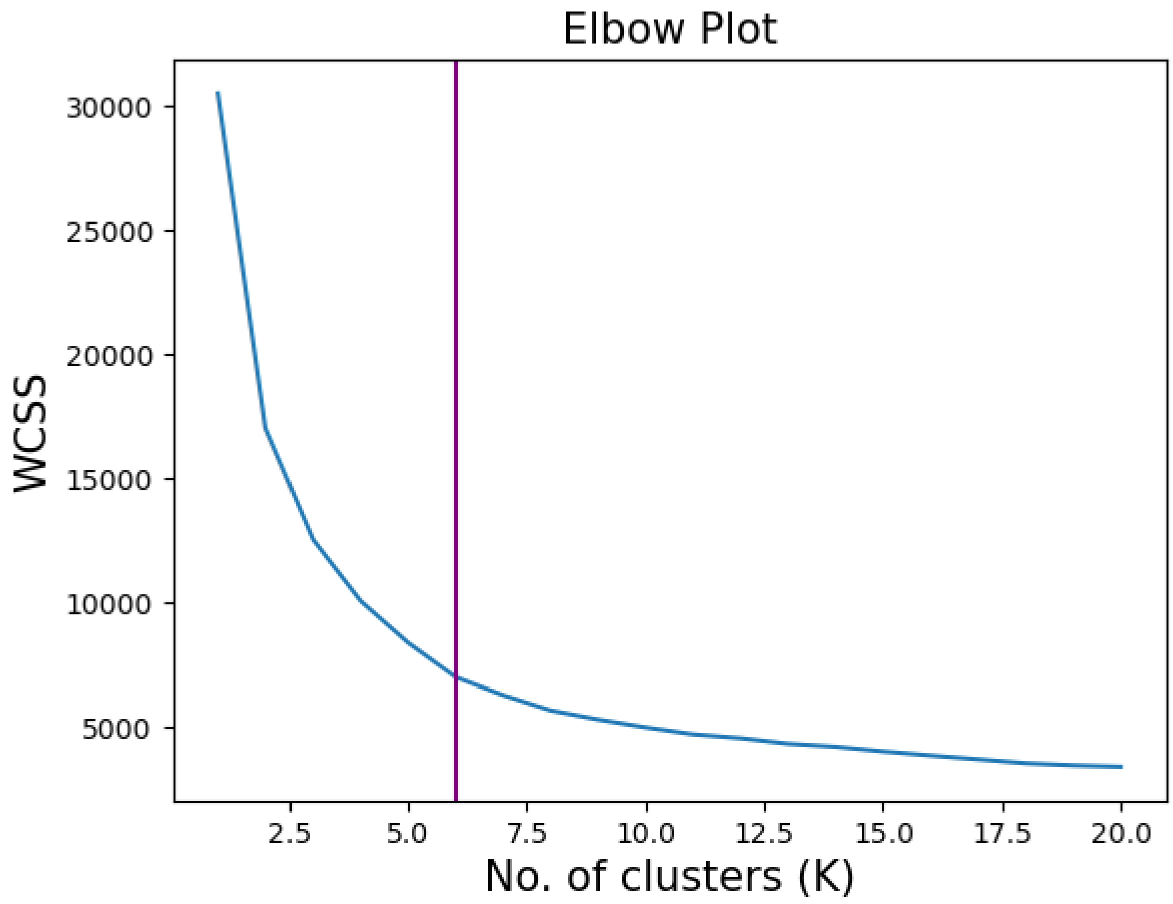
K-Means Clustering

In [78]:
```
wcss  = []

for i in range(1,21):
    kmeans = KMeans(n_clusters = i, random_state = 10)
    kmeans.fit(df_pca)
    wcss.append(kmeans.inertia_)
```

In [79]:
```
plt.plot(range(1,21), wcss)

plt.title('Elbow Plot', fontsize = 15)
plt.xlabel('No. of clusters (K)', fontsize = 15)
plt.ylabel('WCSS', fontsize = 15)

plt.show()
```



In [80]:
```python
plt.plot(range(1,21), wcss)

# set the axes and plot labels
# set the font size using 'fontsize'
plt.title('Elbow Plot', fontsize = 15)
plt.xlabel('No. of clusters (K)', fontsize = 15)
plt.ylabel('WCSS', fontsize = 15)

# plot a vertical line at the elbow
plt.axvline(x = 6, color = 'purple')

# display the plot
plt.show()
```



Interpretation: We can see that the for K = 6, there is an elbow in the plot. Before this elbow point, the WCSS is decreasing rapidly and after K = 6, the WCSS is decreasing slowly.

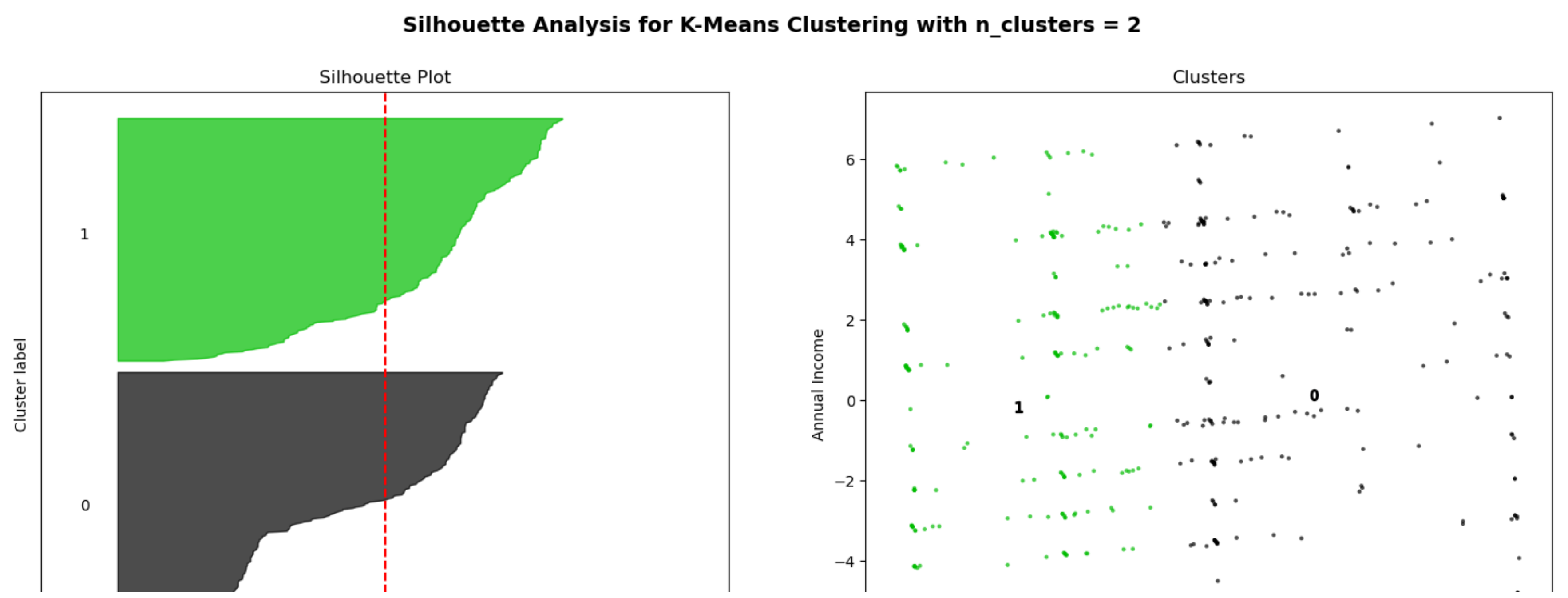
Now, let us use the silhouette score method to identify the optimal value of K.

Optimal Value of K Using Silhouette Score

In [83]:
```python
n_clusters = [2, 3, 4, 5, 6,7]

for K in n_clusters:
    cluster = KMeans (n_clusters= K, random_state= 10)
    predict = cluster.fit_predict(df_pca)
    score = silhouette_score(df_pca, predict, random_state= 10)
    print ("For {} clusters the silhouette score is {})".format(K, score))
```

```
For 2 clusters the silhouette score is 0.3503302858536722)
For 3 clusters the silhouette score is 0.3031171189178978)
For 4 clusters the silhouette score is 0.3156622058518483)
For 5 clusters the silhouette score is 0.2994635218729511)
For 6 clusters the silhouette score is 0.3076228107561279)
For 7 clusters the silhouette score is 0.30251036232888245)
```

In [84]:
```
n_clusters = [2,3,4,5,6,7]
X = np.array(df_pca)

for K in n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    model = KMeans(n_clusters = K, random_state = 10)
    cluster_labels = model.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
    y_lower = 10
    for i in range(K):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / K)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10

    ax1.set_title("Silhouette Plot")
    ax1.set_xlabel("Silhouette coefficient")
    ax1.set_ylabel("Cluster label")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8])
    colors = cm.nipy_spectral(cluster_labels.astype(float) / K)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k')
    centers = model.cluster_centers_

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50, edgecolor='k')

    ax2.set_title("Clusters")
    ax2.set_xlabel("Spending Score")
    ax2.set_ylabel("Annual Income")
    plt.suptitle(("Silhouette Analysis for K-Means Clustering with n_clusters = %d" % K), fontsize=14,fontweight='bolc

plt.show()
```



In [87]:
```
# build a K-Means model with 5 clusters
new_clust = KMeans(n_clusters = 6, random_state = 10)
new_clust.fit(df_pca)
df_pca['Cluster'] = new_clust.labels_
```

In [88]:
```
df_pca.head()
```

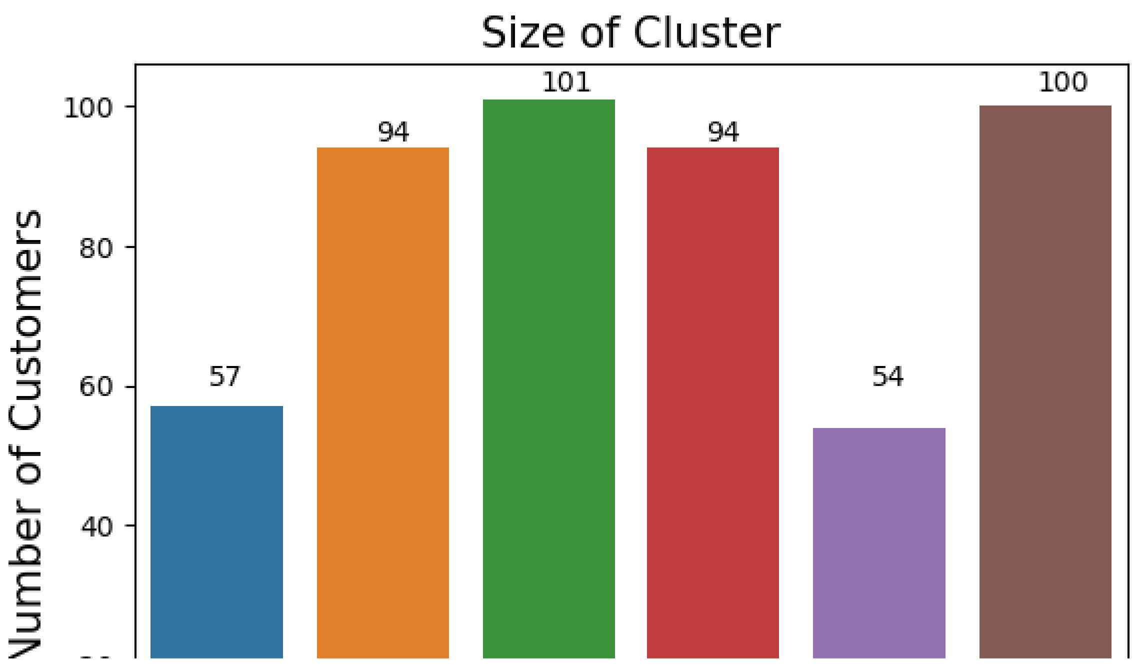
Out[88]:

	PC1	PC2	PC3	PC4	PC5	Cluster
0	-9.262208	-1.246360	-2.567580	0.266860	0.200472	1
1	1.168775	-0.483509	1.756509	-1.088134	1.346846	5
2	1.502789	-5.490850	1.147391	-0.939926	0.756805	5
3	10.736984	2.965870	-2.411270	-0.465814	0.747689	0
4	-0.252974	4.416256	2.389900	1.076906	0.145288	2

In [89]:
```
df_pca.Cluster.value_counts()
```

Out[89]:
```
2    101
5    100
1     94
3     94
0     57
4     54
Name: Cluster, dtype: int64
```

In [95]:
```
sns.countplot(data= df_pca, x = 'Cluster')

plt.title('Size of Cluster', fontsize = 15)
plt.xlabel('Clusters', fontsize = 15)
plt.ylabel('Number of Customers', fontsize = 15)

plt.text(x = -0.05, y =60, s = np.unique(new_clust.labels_, return_counts=True)[1][0])
plt.text(x = 0.95, y =95, s = np.unique(new_clust.labels_, return_counts=True)[1][1])
plt.text(x = 1.95, y =102, s = np.unique(new_clust.labels_, return_counts=True)[1][2])
plt.text(x = 2.95, y =95, s = np.unique(new_clust.labels_, return_counts=True)[1][3])
plt.text(x = 3.95, y =60, s = np.unique(new_clust.labels_, return_counts=True)[1][4])
plt.text(x = 4.95, y =102, s = np.unique(new_clust.labels_, return_counts=True)[1][5])

plt.show()
```



Cluster 2

In [96]:
```
len(df_pca[df_pca['Cluster'] == 0])
```

Out[96]: 57

In [97]:
```
df_pca[df_pca.Cluster==0].describe()
```

Out[97]:

	PC1	PC2	PC3	PC4	PC5	Cluster
count	57.000000	57.000000	57.000000	57.000000	57.000000	57.0
mean	8.754803	3.693102	-0.497815	-0.217525	0.010422	0.0
std	2.466693	1.639907	1.770061	1.446855	1.109417	0.0
min	4.674983	-0.265743	-3.767152	-4.901782	-2.936649	0.0
25%	6.265543	2.667615	-2.173110	-0.731551	-0.453207	0.0
50%	8.838590	3.781217	-0.813832	-0.064762	0.464963	0.0
75%	11.542602	4.962079	0.925016	0.438758	0.590147	0.0
max	11.752290	7.035331	3.276542	4.535248	2.917115	0.0

Cluster 3

In [98]:
```
len(df_pca[df_pca['Cluster'] == 1])
```

Out[98]: 94

In [99]:
```
df_pca[df_pca.Cluster==1].describe()
```

Out[99]:

	PC1	PC2	PC3	PC4	PC5	Cluster
count	94.000000	94.000000	94.000000	94.000000	94.000000	94.0
mean	-6.173217	-3.535887	-0.255052	-0.095540	0.135689	1.0
std	2.451086	1.782500	1.810100	1.303063	0.989583	0.0
min	-9.340700	-6.232069	-3.219197	-2.901430	-3.055702	1.0
25%	-9.051964	-5.834795	-2.329625	-0.946471	-0.110508	1.0
50%	-5.238163	-3.160489	-0.131824	-0.263121	0.298601	1.0
75%	-3.911378	-2.058475	1.062423	0.263193	0.638029	1.0
max	-2.884924	-0.726907	3.299267	4.575728	2.758766	1.0

Cluster 4

In [100]:
```
len(df_pca[df_pca['Cluster'] == 2])
```

Out[100]: 101

In [101]:
```
df_pca[df_pca.Cluster==2].describe()
```

Out[101]:

	PC1	PC2	PC3	PC4	PC5	Cluster
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.0
mean	0.495415	3.364348	0.336561	0.196106	-0.026819	2.0
std	1.691418	1.584346	1.811519	1.401206	1.019940	0.0
min	-3.254616	0.428823	-2.774788	-2.635203	-2.625741	2.0
25%	-0.556506	2.321610	-0.561469	-0.696026	-0.623723	2.0
50%	0.957107	3.389242	0.374655	-0.094560	0.106971	2.0
75%	1.154803	4.444959	1.697557	0.842354	0.402824	2.0
max	4.419544	6.589606	3.579641	4.800258	2.821771	2.0

Cluster 5

In [102]:
```
len(df_pca[df_pca['Cluster'] == 3])
```

Out[102]: 94

In [103]:
```
df_pca[df_pca.Cluster==3].describe()
```

Out[103]:

	PC1	PC2	PC3	PC4	PC5	Cluster
count	94.000000	94.000000	94.000000	94.000000	94.000000	94.0
mean	-6.730059	2.896565	-0.218283	-0.226355	-0.107398	3.0
std	2.570382	1.805925	1.616934	1.034805	0.950654	0.0
min	-9.825877	-0.225199	-2.699397	-1.597796	-3.043514	3.0
25%	-9.491667	1.130301	-1.426663	-0.884685	-0.656578	3.0
50%	-5.470635	2.625061	0.011388	-0.494429	0.177475	3.0
75%	-4.288576	4.135418	1.162170	0.317419	0.517612	3.0
max	-3.176419	6.180595	2.625427	4.469229	2.050280	3.0

Cluster 6

In [104]:
```
len(df_pca[df_pca['Cluster'] == 4])
```

Out[104]: 54

In [105]:
```
df_pca[df_pca.Cluster==4].describe()
```

Out[105]:

	PC1	PC2	PC3	PC4	PC5	Cluster
count	54.000000	54.000000	54.000000	54.000000	54.000000	54.0
mean	10.177791	-3.808935	-0.152446	-0.361505	0.108067	4.0
std	2.417262	1.702287	1.672347	1.054360	1.174297	0.0
min	5.817402	-5.319577	-2.391329	-3.533300	-2.519543	4.0
25%	7.230519	-4.991398	-1.950171	-1.063927	-0.343579	4.0
50%	11.822600	-4.876219	-0.211746	-0.333412	0.480868	4.0
75%	12.031493	-2.877350	1.116334	0.270227	0.616434	4.0
max	12.155694	0.078102	3.368880	2.553171	3.058939	4.0

Cluster 7

In [106]:
```
len(df_pca[df_pca['Cluster'] == 5])
```

Out[106]: 100

In [107]:
```
df_pca[df_pca.Cluster==5].describe()
```

Out[107]:

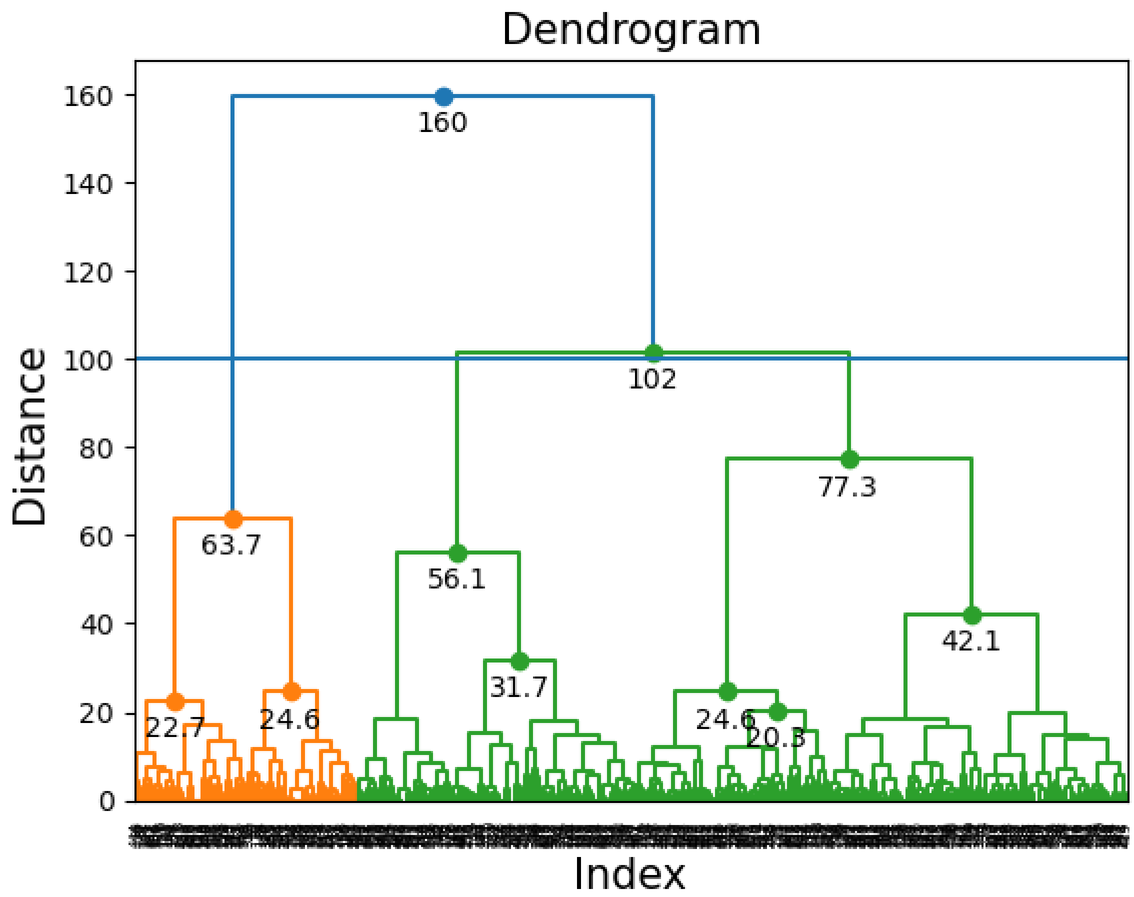
	PC1	PC2	PC3	PC4	PC5	Cluster
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.0
mean	1.142465	-2.845273	0.471084	0.423716	-0.063803	5.0
std	1.832966	2.028479	1.731743	1.734804	1.157244	0.0
min	-2.961083	-5.896807	-2.592574	-2.465212	-2.864044	5.0
25%	0.323304	-5.498824	-0.907223	-0.785322	-0.791874	5.0
50%	1.365484	-2.571362	0.534035	-0.100642	0.059009	5.0
75%	1.736747	-0.847540	1.604805	1.576395	0.624910	5.0
max	6.034595	0.601928	4.912580	6.264353	3.436725	5.0

Hierarchical Clustering

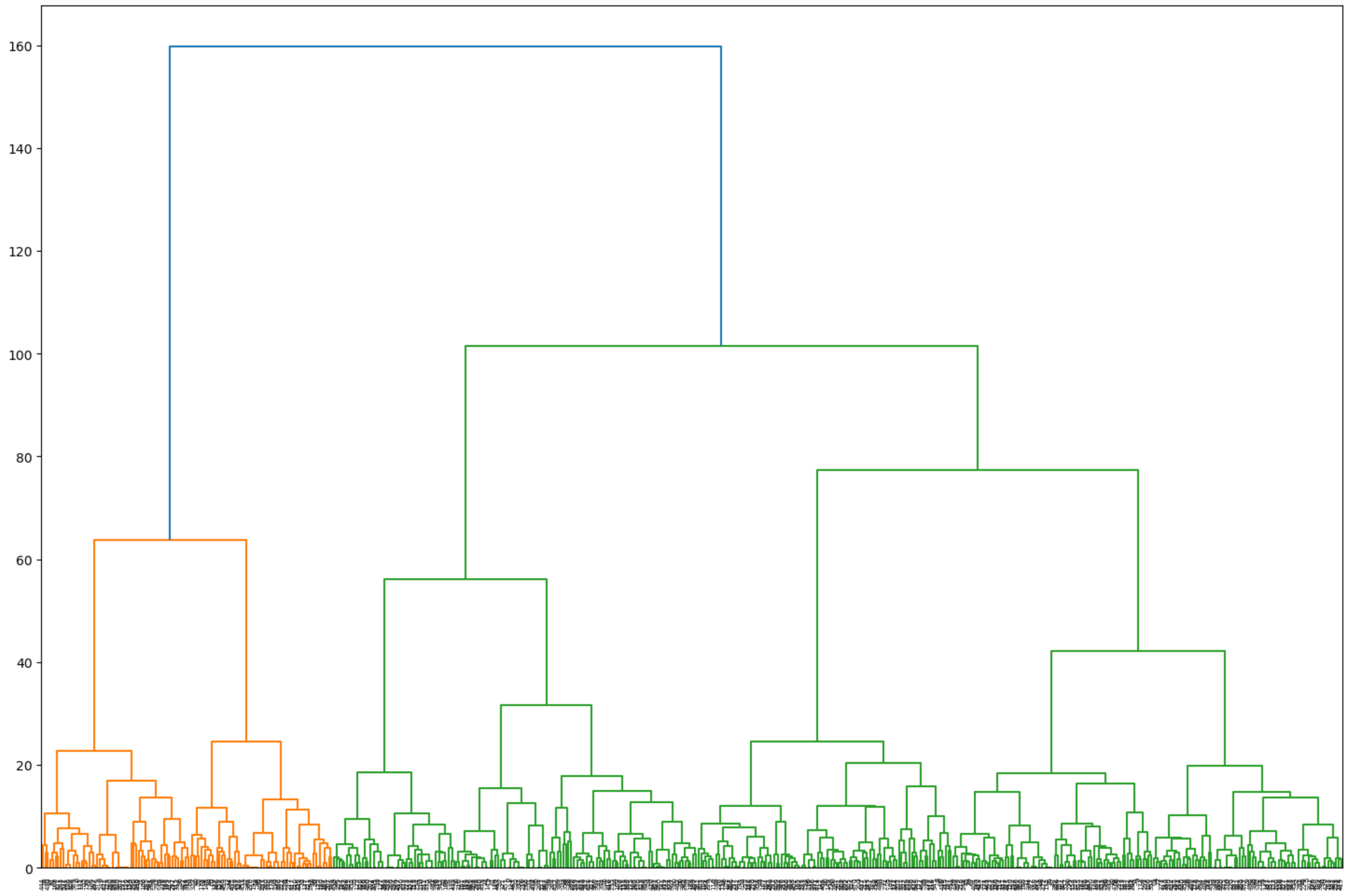
In [109]:
```
link_mat = linkage(df_pca, method = 'ward')

# print first 10 observations of the linkage matrix 'link_mat'
print(link_mat[0:10])
```

```
[[129. 290.   0.   2.]
 [431. 500.   0.   3.]
 [ 97. 320.   0.   2.]
 [370. 502.   0.   3.]
 [375. 503.   0.   4.]
 [ 83. 283.   0.   2.]
 [280. 460.   0.   2.]
 [ 68. 211.   0.   2.]
 [397. 507.   0.   3.]
 [371. 451.   0.   2.]]
```

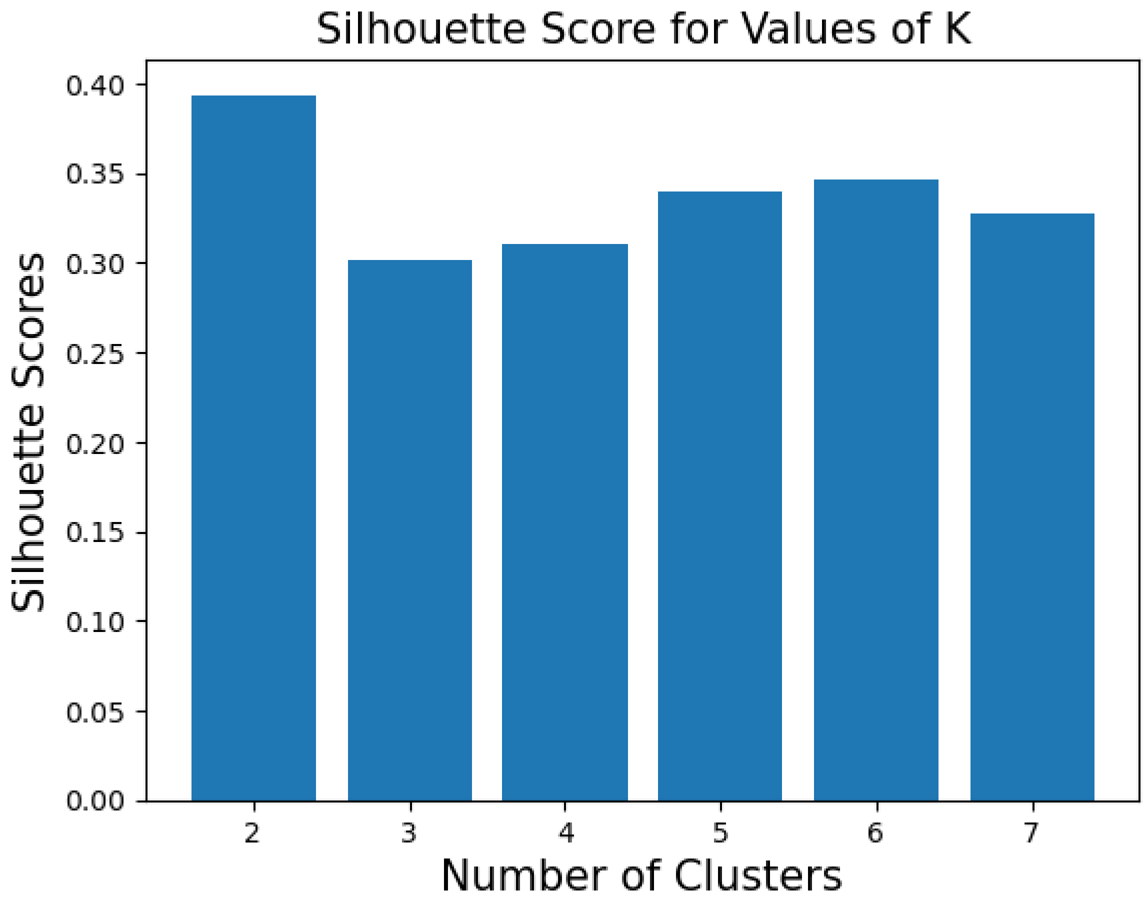
In [110]:
```
dendro = dendrogram(link_mat)
for i, d, c in zip(dendro['icoord'], dendro['dcoord'], dendro['color_list']):
    x = sum(i[1:3])/2
    y = d[1]
    if y > 20:
        plt.plot(x, y, 'o', c=c)
        plt.annotate("%.3g" % y, (x, y), xytext=(0, -5), textcoords='offset points', va='top', ha='center')
plt.axhline(y = 100)
plt.title('Dendrogram', fontsize = 15)
plt.xlabel('Index', fontsize = 15)
plt.ylabel('Distance', fontsize = 15)
plt.show()
```



In [111]:
```
plt.figure(figsize = (18,12))
dendrogram(link_mat)
plt.show()
```



In [114]:
```
K = [2,3,4,5,6,7]

silhouette_scores = []
for i in K:
    model = AgglomerativeClustering(n_clusters = i)
    silhouette_scores.append(silhouette_score(df_pca, model.fit_predict(df_pca)))
plt.bar(K, silhouette_scores)

plt.title('Silhouette Score for Values of K', fontsize = 15)
plt.xlabel('Number of Clusters', fontsize = 15)
plt.ylabel('Silhouette Scores', fontsize = 15)

plt.show()
```



In [115]:
```
clusters = AgglomerativeClustering(n_clusters=2, linkage='ward')

clusters.fit(df_pca)
```

Out[115]:
```
AgglomerativeClustering()
```

In [116]:
```
df_pca['Agg Cluster'] = clusters.labels_

df_pca.head()
```

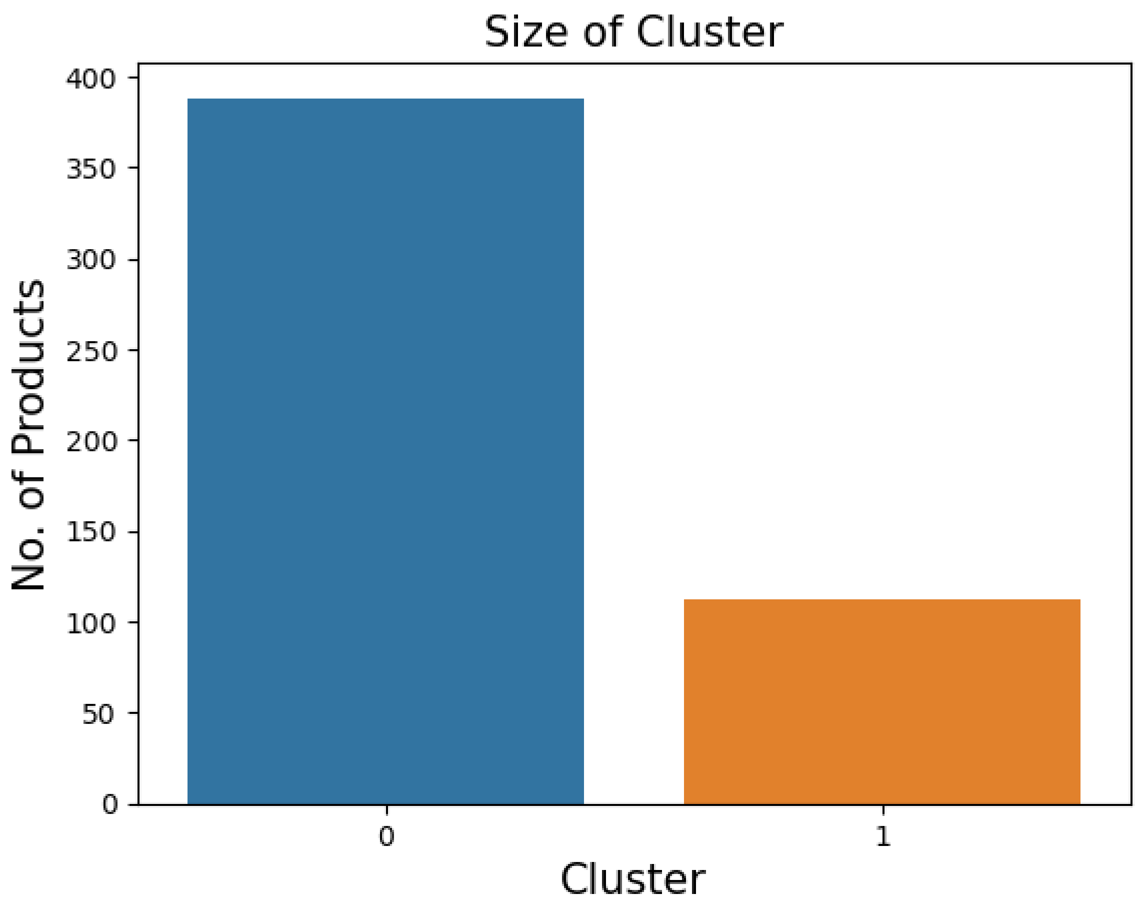
Out[116]:

	PC1	PC2	PC3	PC4	PC5	Cluster	Agg Cluster
0	-9.262208	-1.246360	-2.567580	0.266860	0.200472	1	0
1	1.168775	-0.483509	1.756509	-1.088134	1.346846	5	0
2	1.502789	-5.490850	1.147391	-0.939926	0.756805	5	0
3	10.736984	2.965870	-2.411270	-0.465814	0.747689	0	1
4	-0.252974	4.416256	2.389900	1.076906	0.145288	2	0

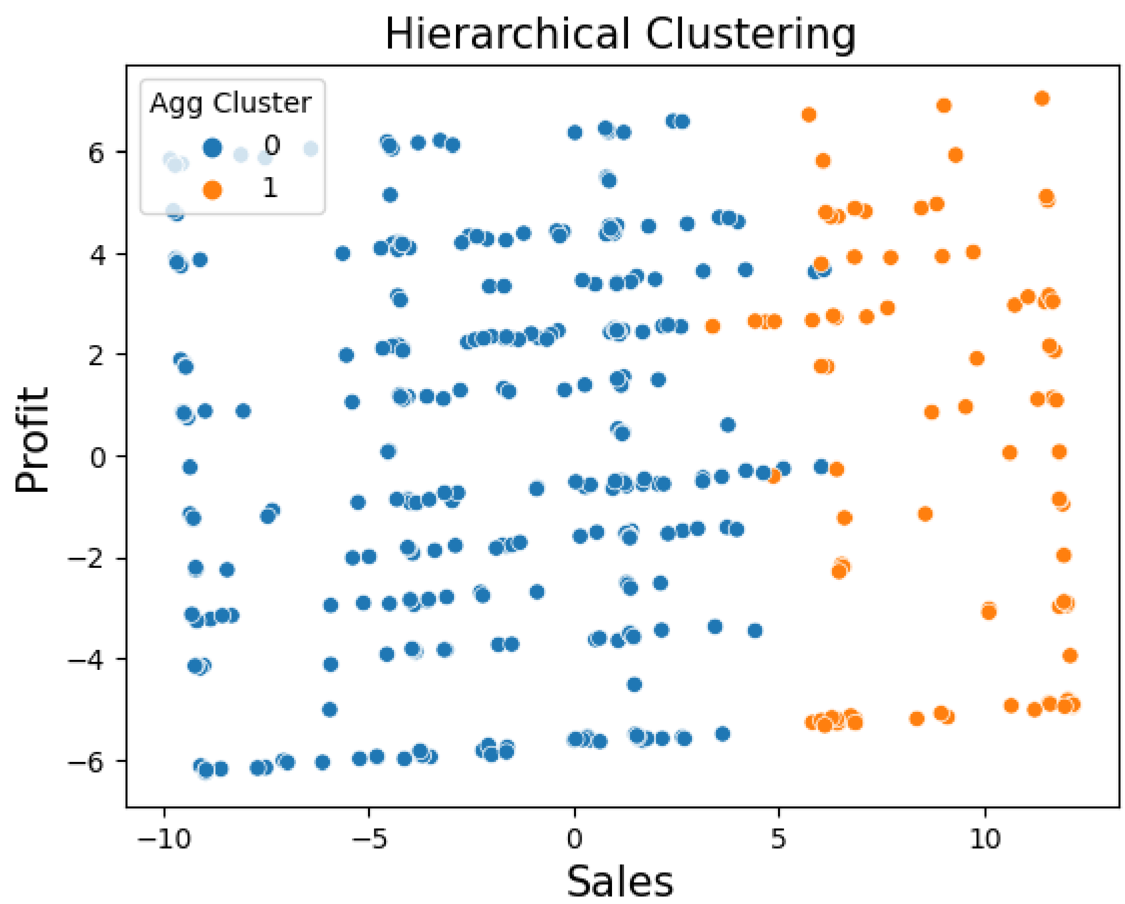
In [117]:
```
df_pca['Agg Cluster'].value_counts()
```

Out[117]:
```
0    388
1    112
Name: Agg Cluster, dtype: int64
```

In [119]:
```
sns.countplot(data = df_pca, x = 'Agg Cluster')

plt.title('Size of Cluster', fontsize = 15)
plt.xlabel('Cluster', fontsize = 15)
plt.ylabel('No. of Products', fontsize = 15)

plt.show()
```



In [120]:
```
# plot the scatterplot to visualize the clusters
sns.scatterplot(x = 'PC1', y = 'PC2', data = df_pca, hue = 'Agg Cluster')

plt.title('Hierarchical Clustering', fontsize = 15)
plt.xlabel('Sales', fontsize = 15)
plt.ylabel('Profit', fontsize = 15)

# display the plot
plt.show()
```



Analysis of Cluster_1

In [123]:
```
df_pca['Agg Cluster'].value_counts()[0]
```

Out[123]: 388

In [128]:
```
df_pca[df_pca['Agg Cluster'] == 0].head(10)
```

Out[128]:

	PC1	PC2	PC3	PC4	PC5	Cluster	Agg Cluster
0	-9.262208	-1.246360	-2.567580	0.266860	0.200472	1	0
1	1.168775	-0.483509	1.756509	-1.088134	1.346846	5	0
2	1.502789	-5.490850	1.147391	-0.939926	0.756805	5	0
4	-0.252974	4.416256	2.389900	1.076906	0.145288	2	0
5	-4.183824	1.122334	-0.425883	-0.483475	0.784353	3	0
7	-9.067464	-6.122137	2.268556	-1.495790	0.145169	1	0
8	4.421797	-3.448411	1.502602	4.871930	-2.334254	5	0
9	0.270670	1.392619	1.537734	2.137315	1.536218	2	0
10	0.933944	4.448672	0.233446	-0.561480	-0.044376	2	0
11	1.109854	2.392100	-2.381492	0.361236	0.378332	2	0

In [129]:
```
df_pca[df_pca['Agg Cluster'] == 0].describe()
```

Out[129]:

	PC1	PC2	PC3	PC4	PC5	Cluster	Agg Cluster
count	388.000000	388.000000	388.000000	388.000000	388.000000	388.000000	388.0
mean	-2.704420	-0.006016	0.111227	0.102775	-0.029595	2.755155	0.0
std	4.242743	3.660922	1.762270	1.438359	1.028689	1.495678	0.0
min	-9.825877	-6.232069	-3.219197	-2.901430	-3.055702	0.000000	0.0
25%	-5.021935	-3.127874	-1.354316	-0.823078	-0.608055	2.000000	0.0
50%	-2.657257	0.080340	0.290084	-0.261956	0.186674	2.000000	0.0
75%	1.079361	3.382530	1.464524	0.382302	0.574676	5.000000	0.0
max	6.098243	6.589606	4.912580	6.264353	3.436725	5.000000	0.0

In [130]:
```
df_pca[df_pca['Agg Cluster'] == 0].index.value_counts()
```

Out[130]:
```
0      1
326    1
354    1
353    1
352    1
      ..
172    1
170    1
169    1
167    1
499    1
Length: 388, dtype: int64
```

Analysis of Cluster_2

In [131]:
```
df_pca['Agg Cluster'].value_counts()[1]
```

Out[131]: 112

In [132]:
```
df_pca[df_pca['Agg Cluster'] == 1].head(10)
```

Out[132]:

	PC1	PC2	PC3	PC4	PC5	Cluster	Agg Cluster
3	10.736984	2.965870	-2.411270	-0.465814	0.747689	0	1
6	6.234561	4.746177	-0.984623	-0.050082	0.329282	0	1
12	6.469278	-5.266479	-0.797090	-0.281165	0.260791	4	1
15	6.386913	2.715617	-2.255007	0.383144	0.474162	0	1
17	12.149456	-4.937999	-1.950171	0.270227	0.606786	4	1
21	11.647354	2.092886	-0.046428	-0.417751	1.071840	0	1
23	11.906714	-0.949432	-2.039346	0.337640	0.588389	4	1
32	12.117289	-4.857791	1.147955	-0.900082	-1.775050	4	1
34	10.618978	0.055190	2.529051	-0.166829	1.372932	4	1
36	9.725568	4.012831	-0.275711	-0.064820	-1.843556	0	1

In [133]:
```
df_pca[df_pca['Agg Cluster'] == 1].describe()
```

Out[133]:

	PC1	PC2	PC3	PC4	PC5	Cluster	Agg Cluster
count	112.000000	112.000000	112.000000	112.000000	112.000000	112.000000	112.0
mean	9.368884	0.020842	-0.385322	-0.356040	0.102524	2.008929	1.0
std	2.625180	4.084796	1.731806	1.147831	1.149352	2.002231	0.0
min	3.387603	-5.319577	-3.767152	-4.901782	-2.936649	0.000000	1.0
25%	6.468852	-4.861597	-1.971005	-1.030685	-0.398739	0.000000	1.0
50%	10.365755	0.465917	-0.576058	-0.203263	0.469562	2.000000	1.0
75%	11.821513	3.811919	1.006243	0.375558	0.614342	4.000000	1.0
max	12.155694	7.035331	3.368880	2.553171	3.058939	5.000000	1.0

In [134]:
```
df_pca[df_pca['Agg Cluster'] == 1].index.value_counts()
```

Out[134]:
```
3      1
6      1
323    1
321    1
317    1
      ..
130    1
129    1
126    1
121    1
495    1
Length: 112, dtype: int64
```

In [ ]:

In [ ]: